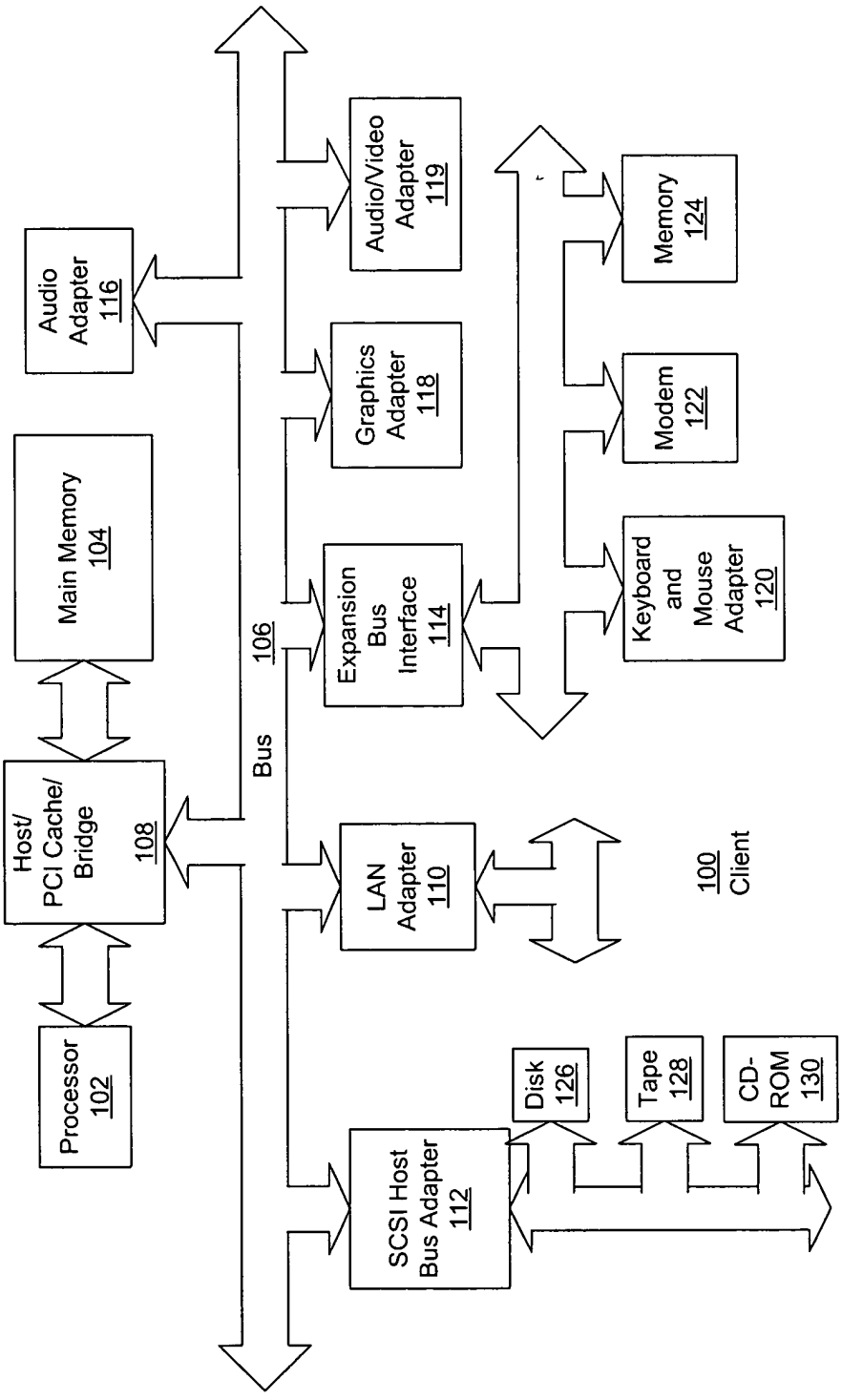

Figure 1
(PRIOR ART)

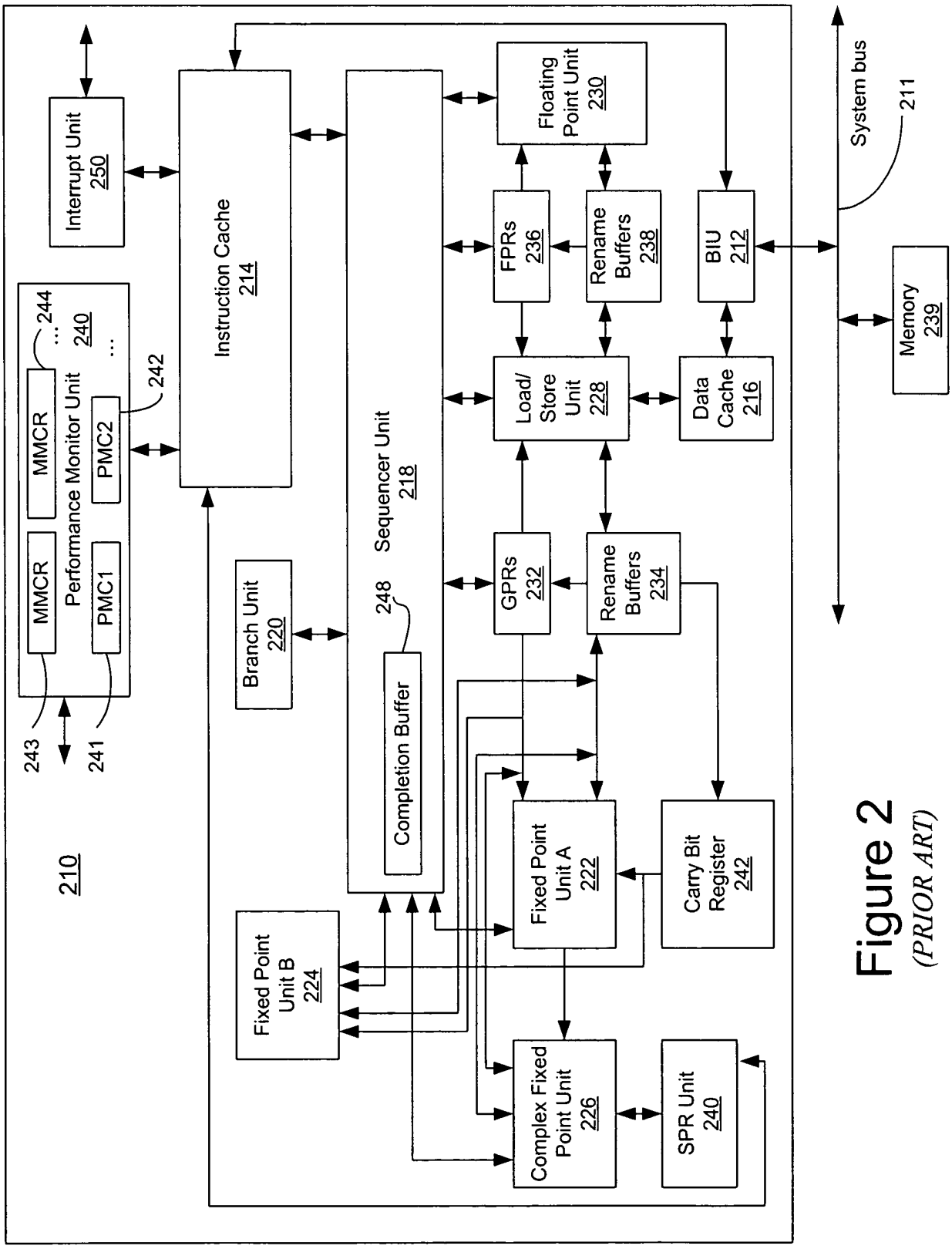

Figure 2
(PRIOR ART)

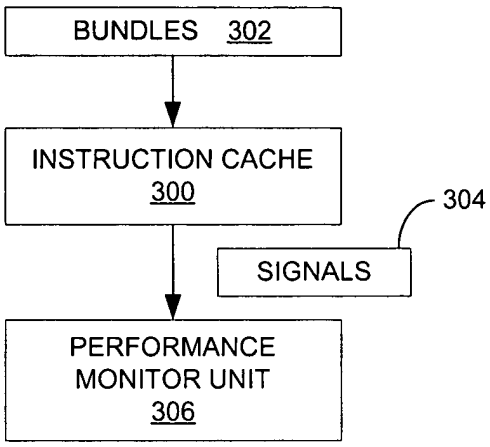

Figure 3

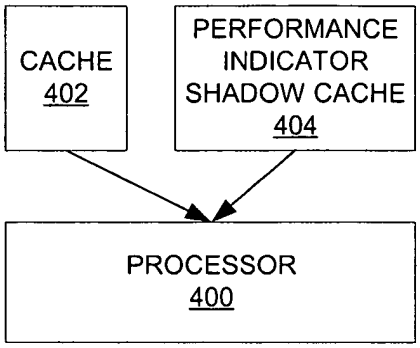

Figure 4

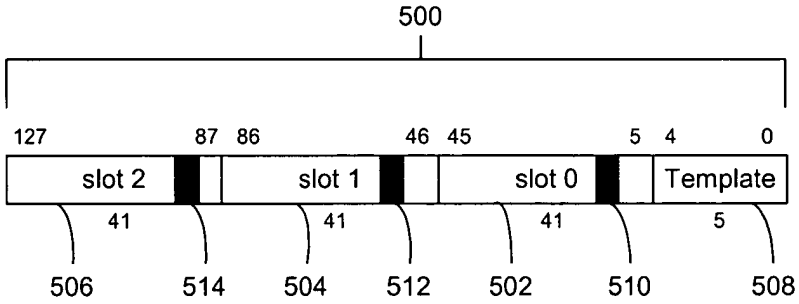

Figure 5

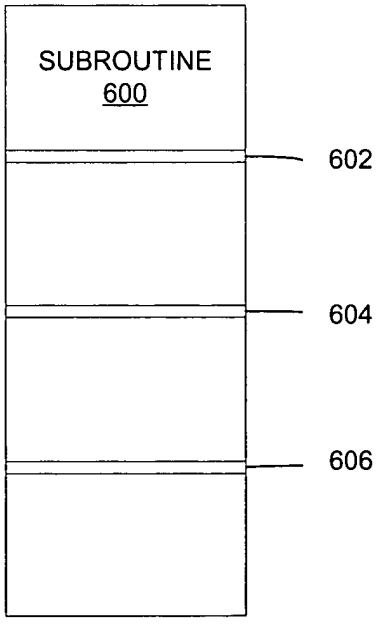

Figure 6A

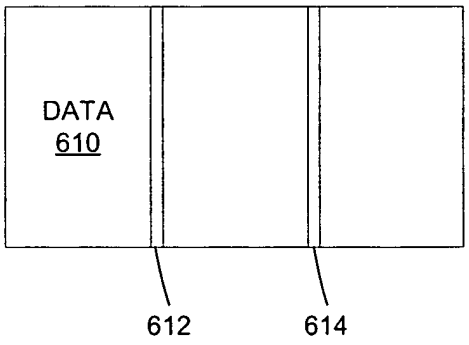

Figure 6B

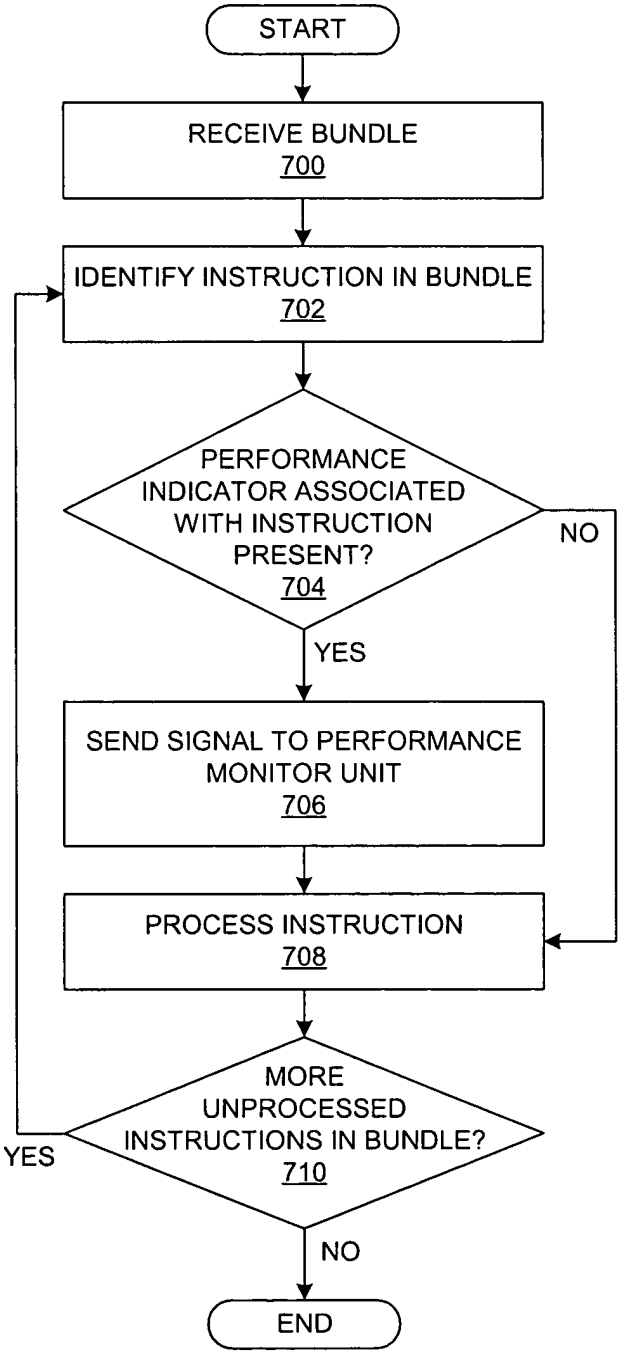

Figure 7

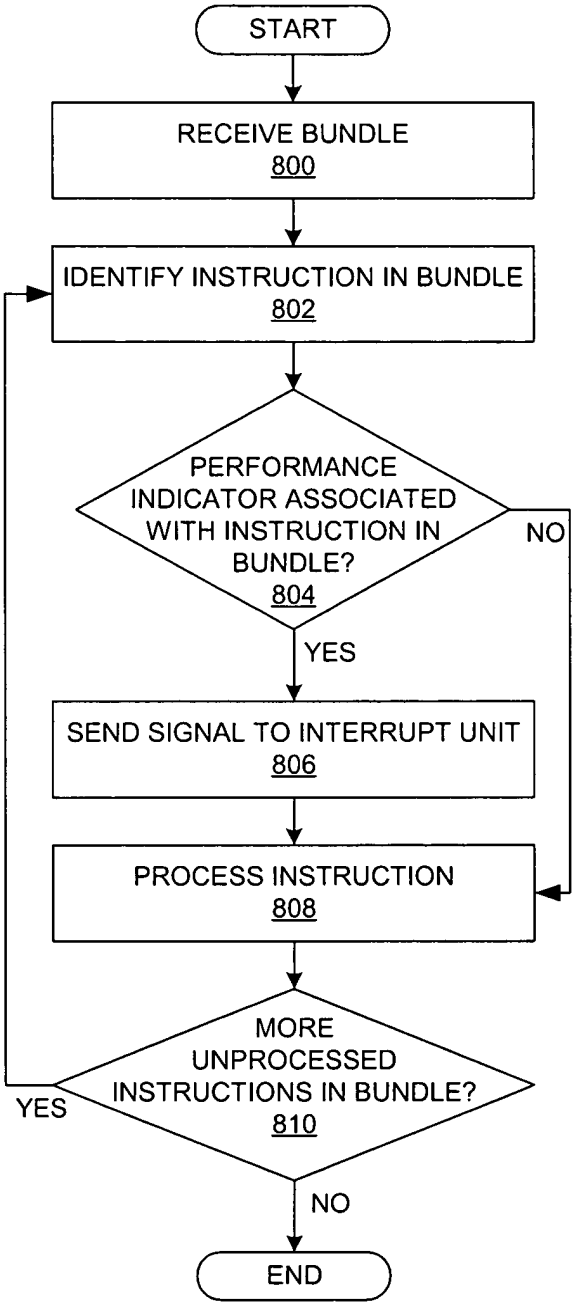

Figure 8

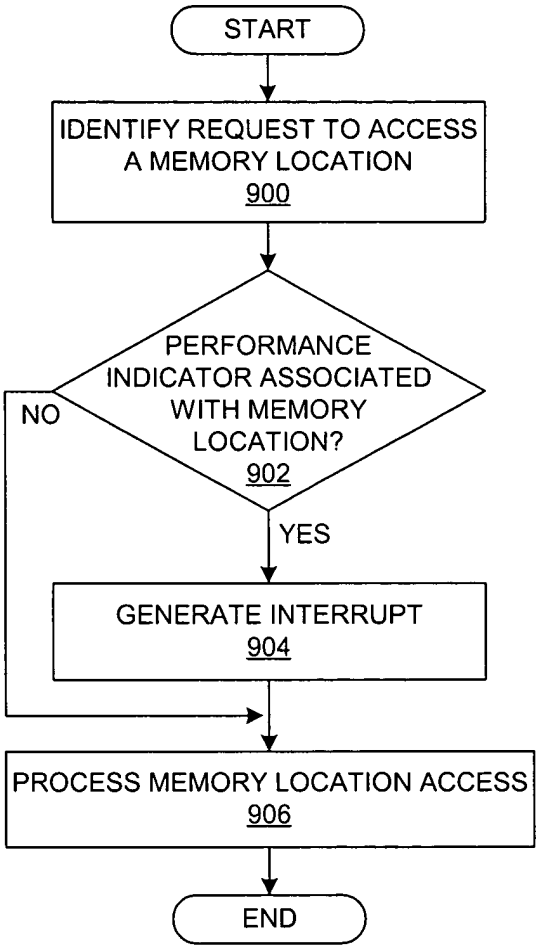

Figure 9

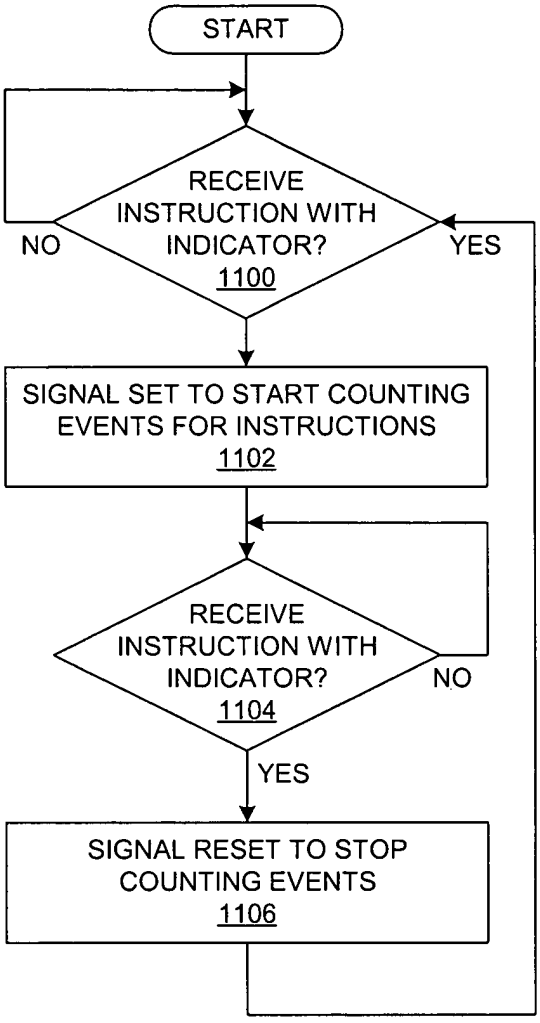

Figure 11

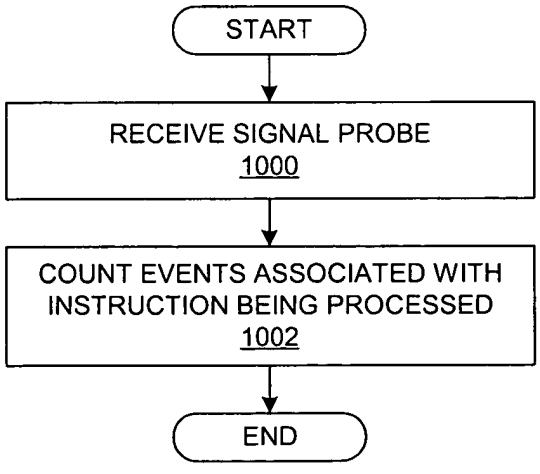

Figure 10

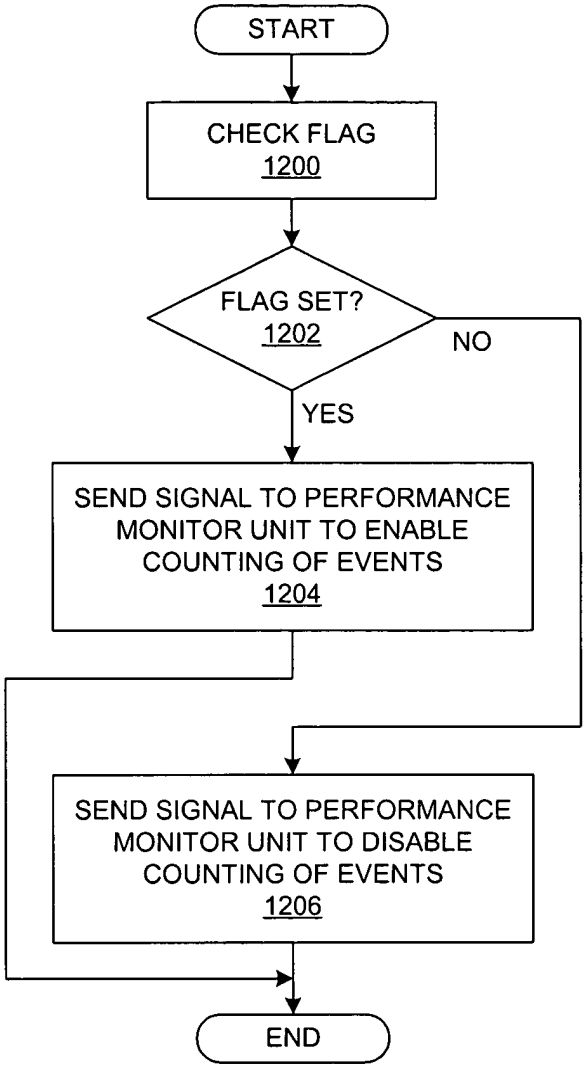

Figure 12

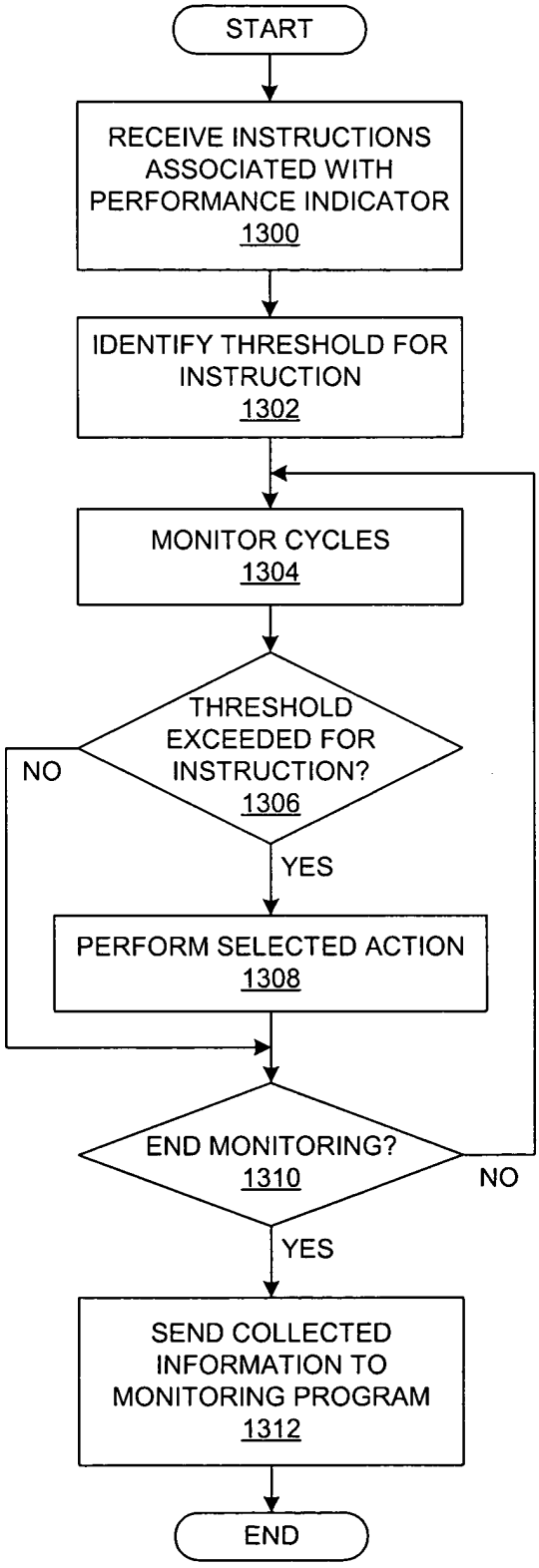

Figure 13

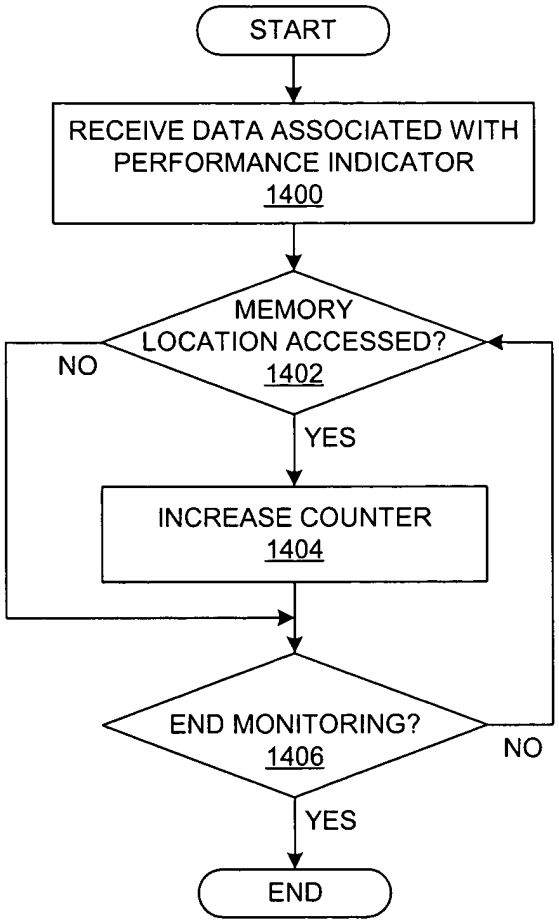
START
RECEIVE DATA ASSOCIATED WITH PERFORMANCE INDICATOR
1400
MEMORY LOCATION ACCESSED?
1402
NO
YES
INCREASE COUNTER
1404
END MONITORING?
1406
NO
YES
END

Figure 14

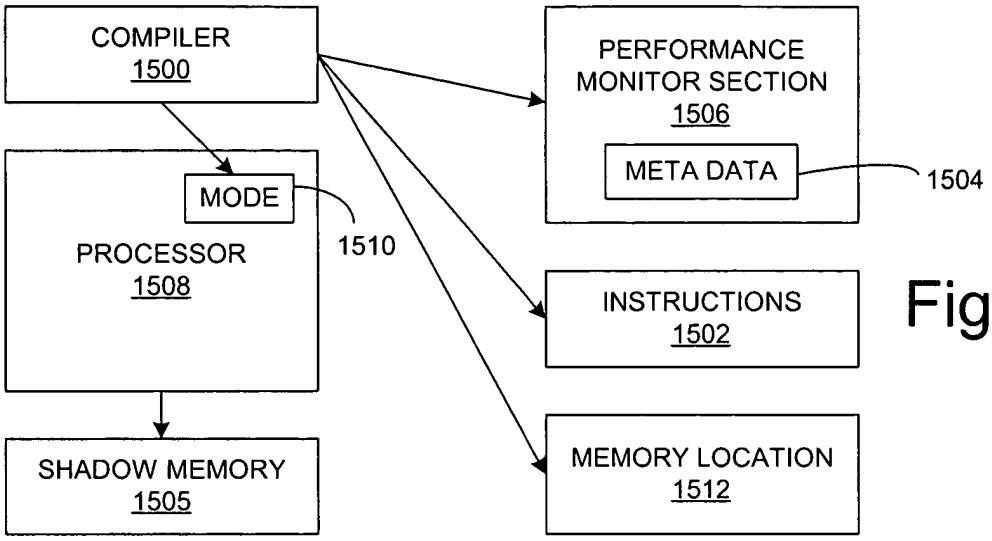

Figure 15

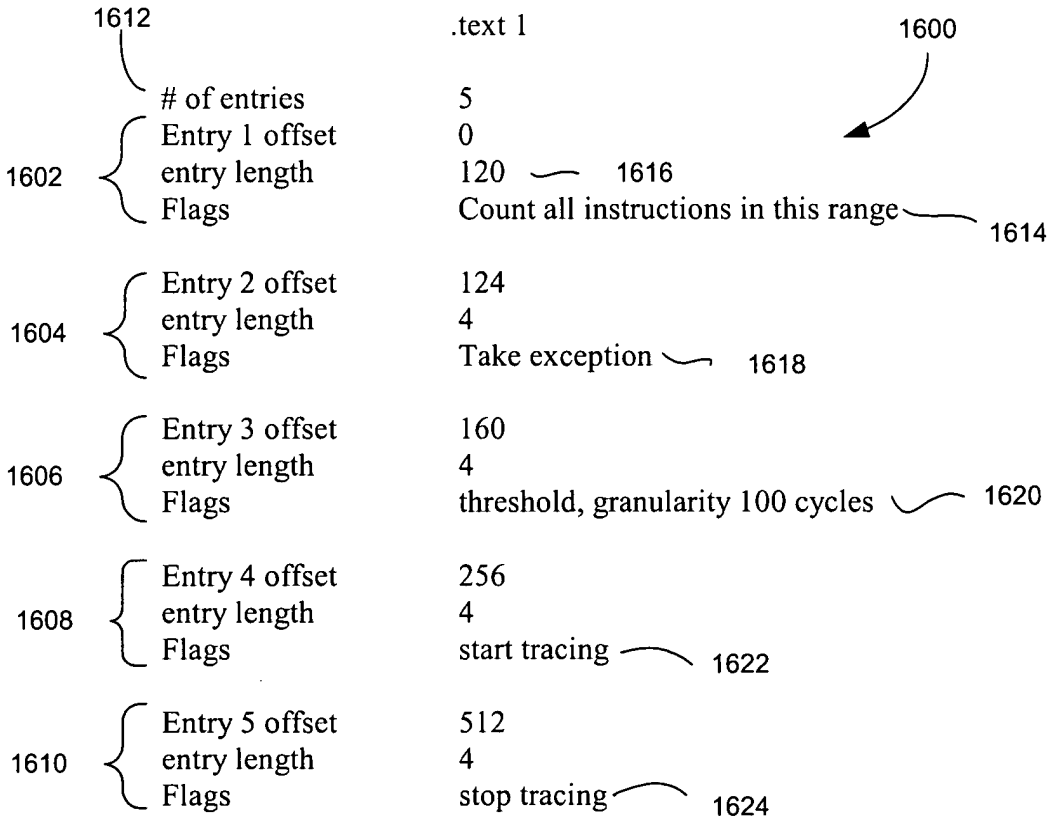

Figure 16

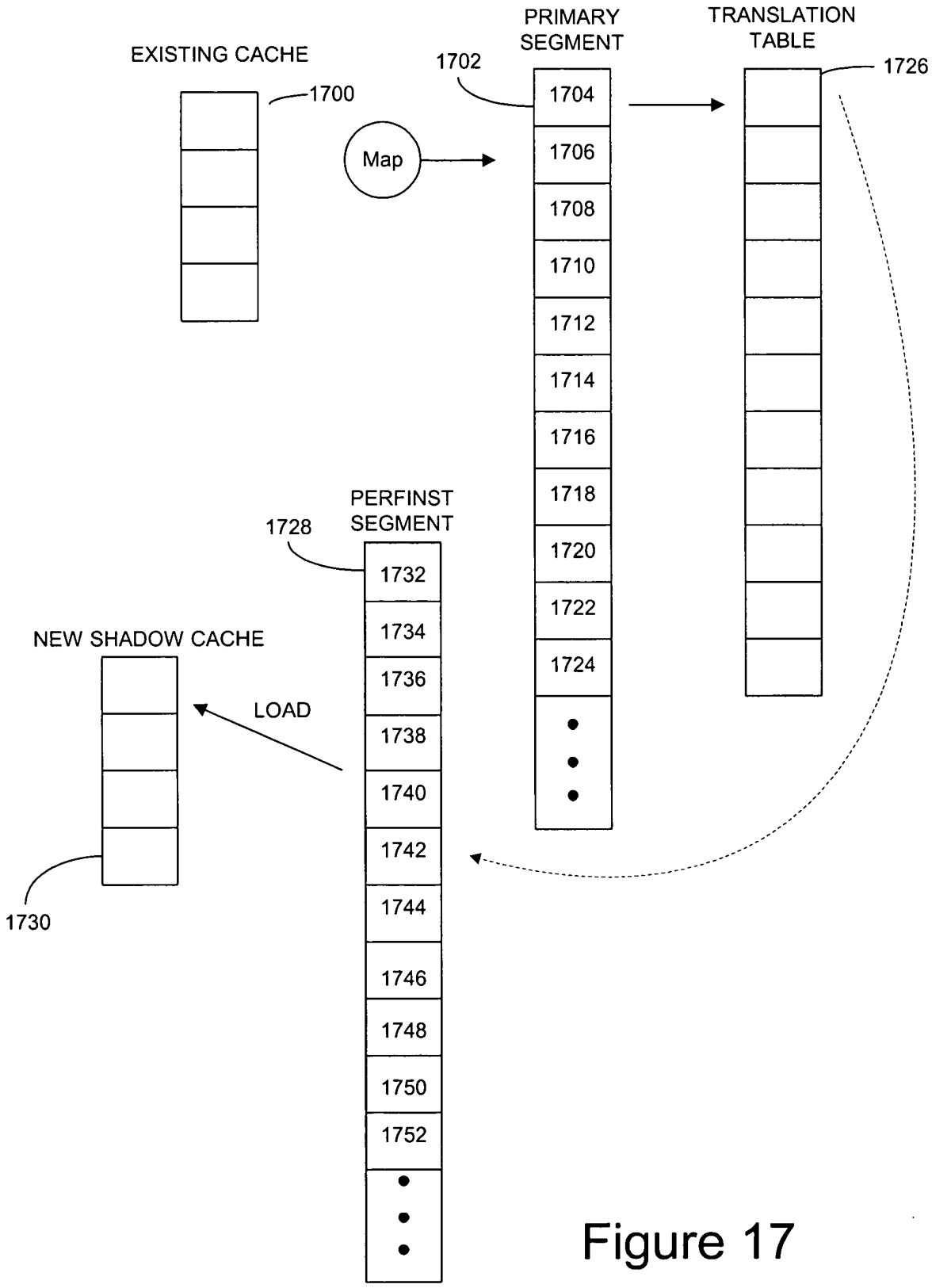
EXISTING CACHE
1700
Map
1702
PRIMARY SEGMENT
1704
1706
1708
1710
1712
1714
1716
1718
1720
1722
1724
TRANSLATION TABLE
1726
PERFINST SEGMENT
1728
1732
1734
1736
1738
1740
1742
1744
1746
1748
1750
1752
NEW SHADOW CACHE
LOAD
1730

Figure 17

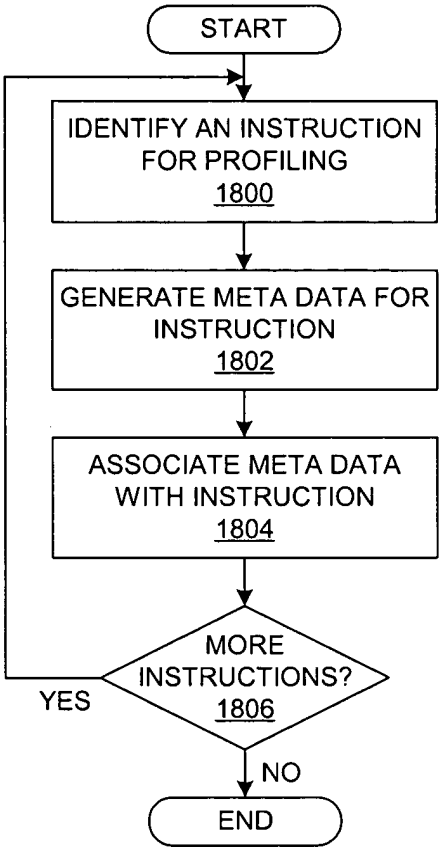

Figure 18

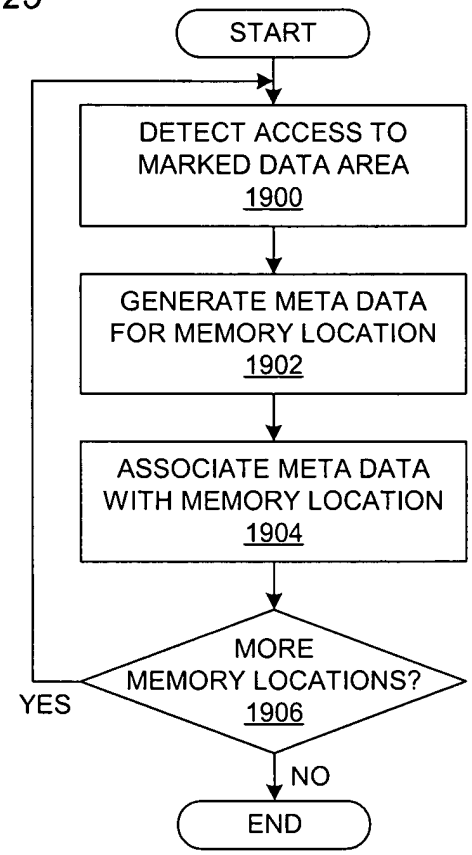

Figure 19

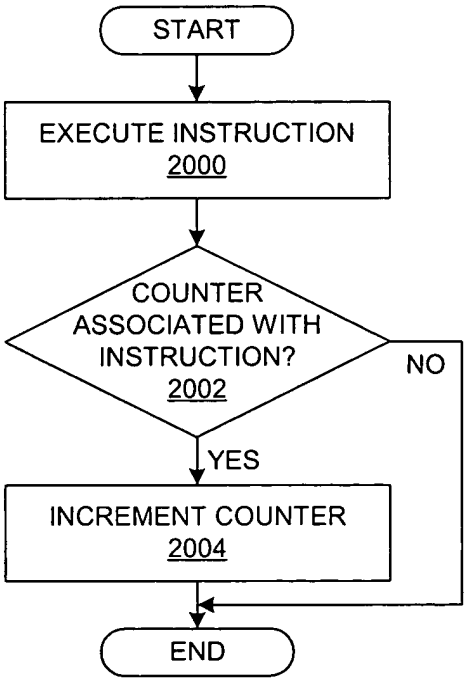

Figure 20

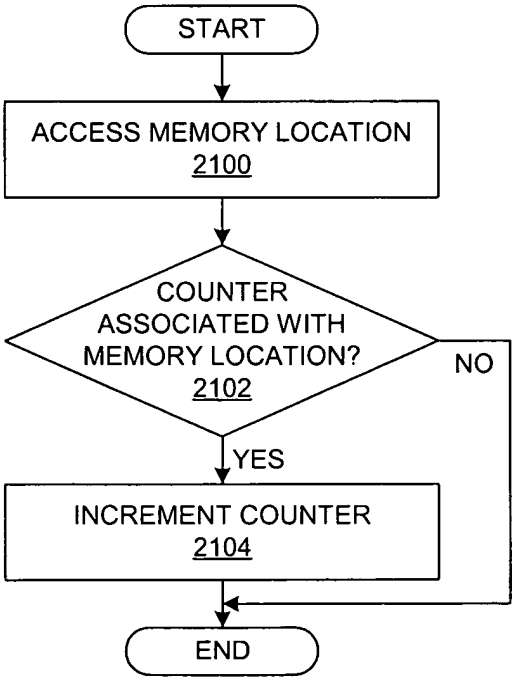

Figure 21

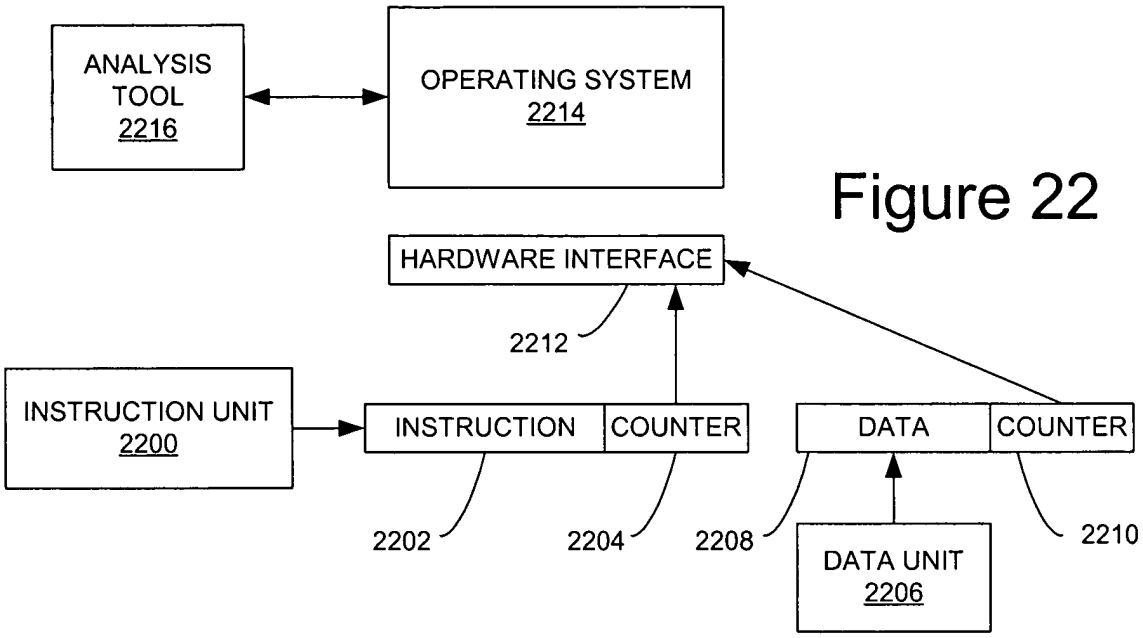

Figure 22

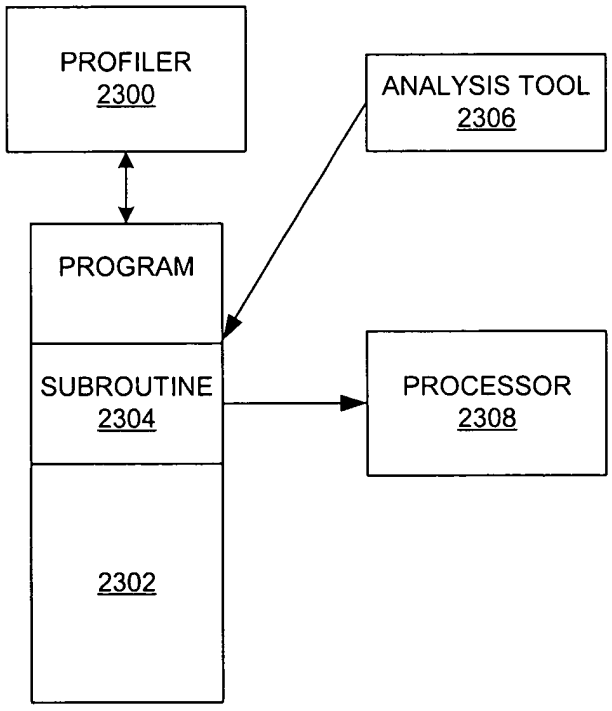

Figure 23

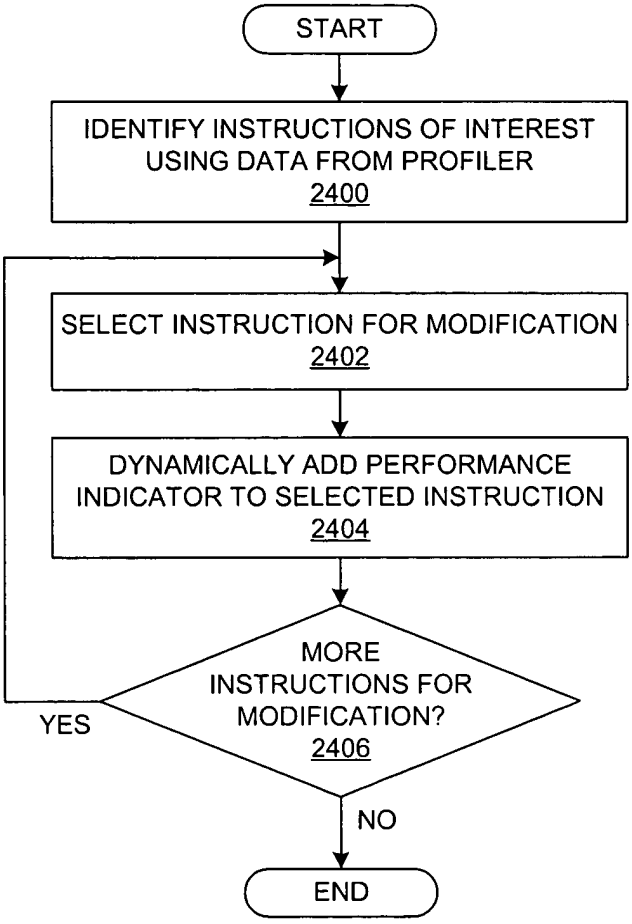

Figure 24

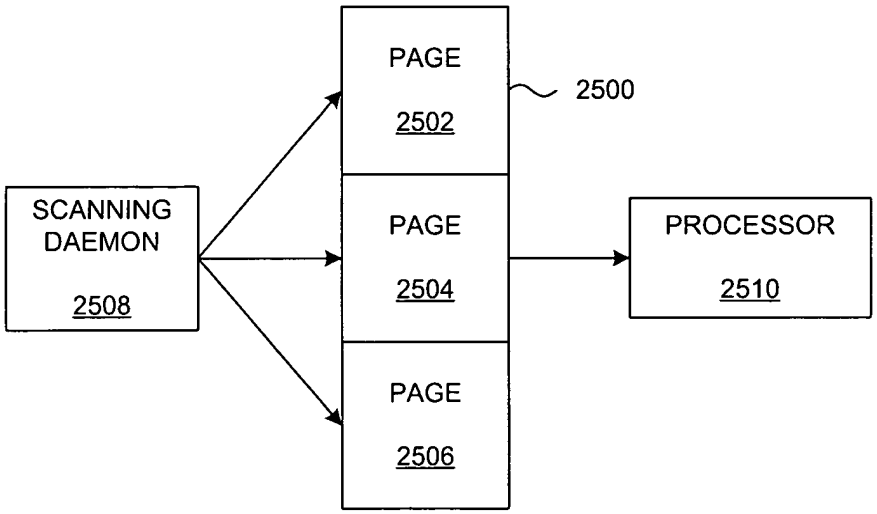

Figure 25

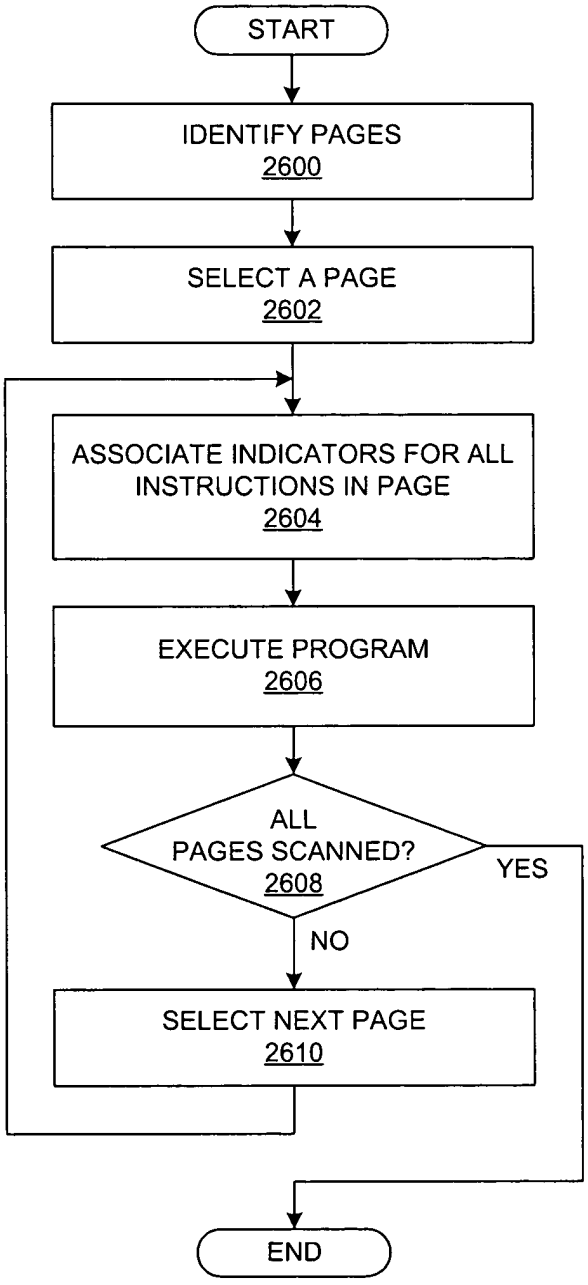

Figure 26

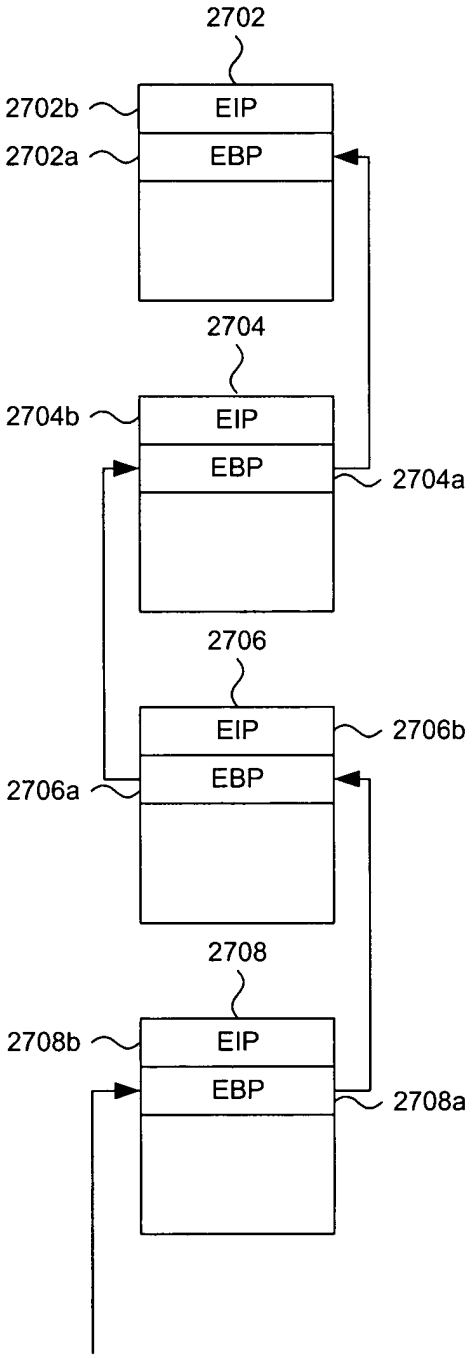

Figure 27

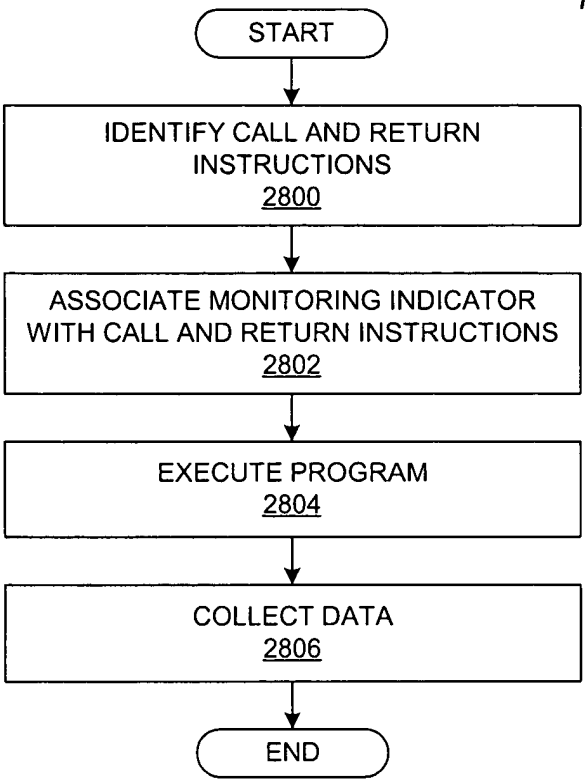

FIGURE 28

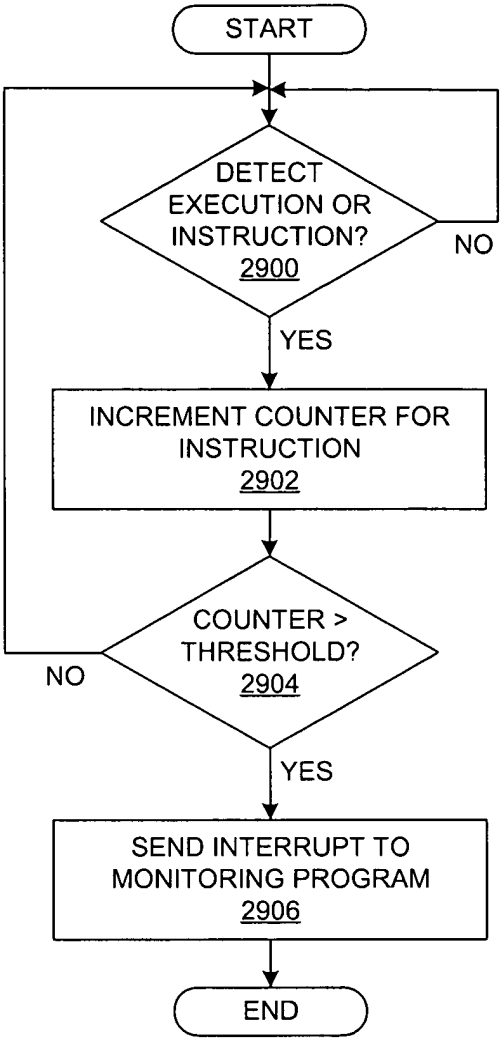

FIGURE 29

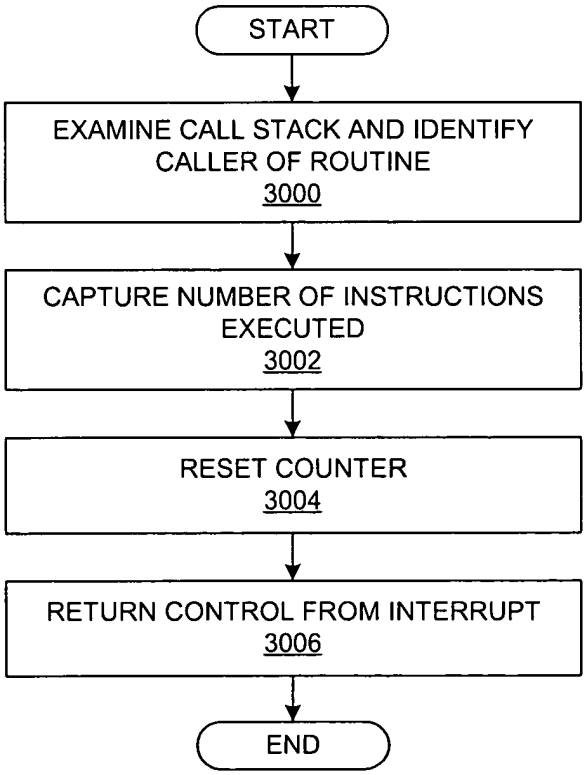

FIGURE 30

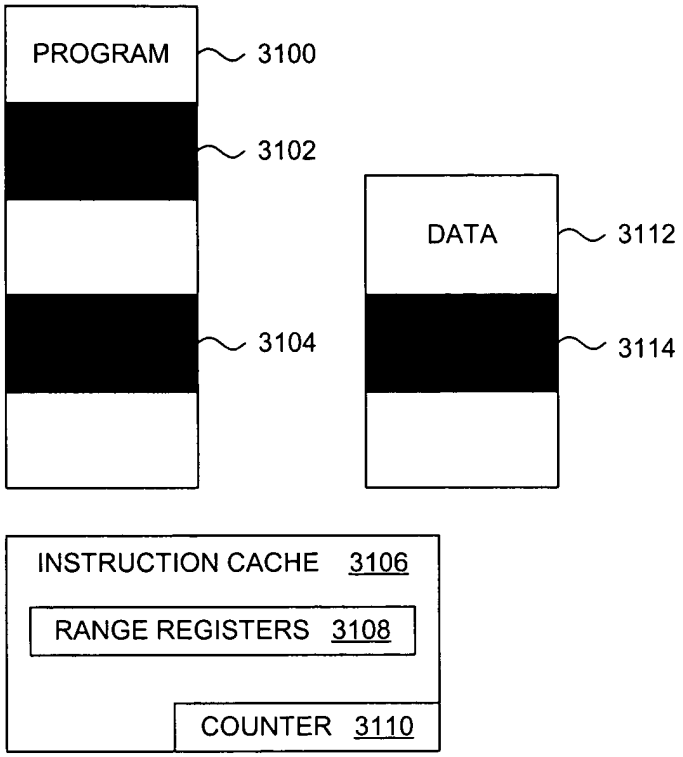

FIGURE 31

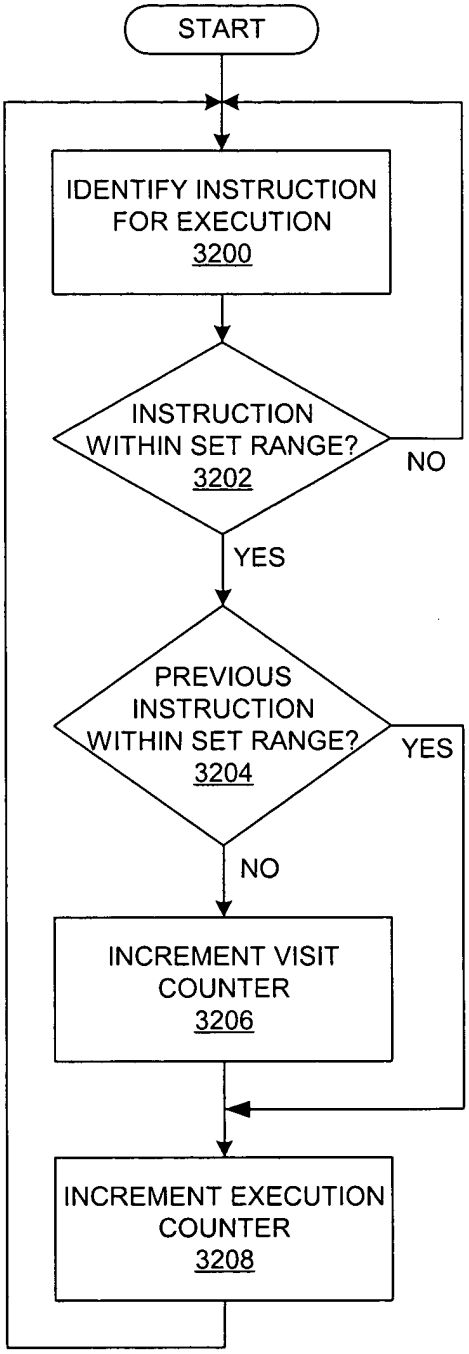

FIGURE 32

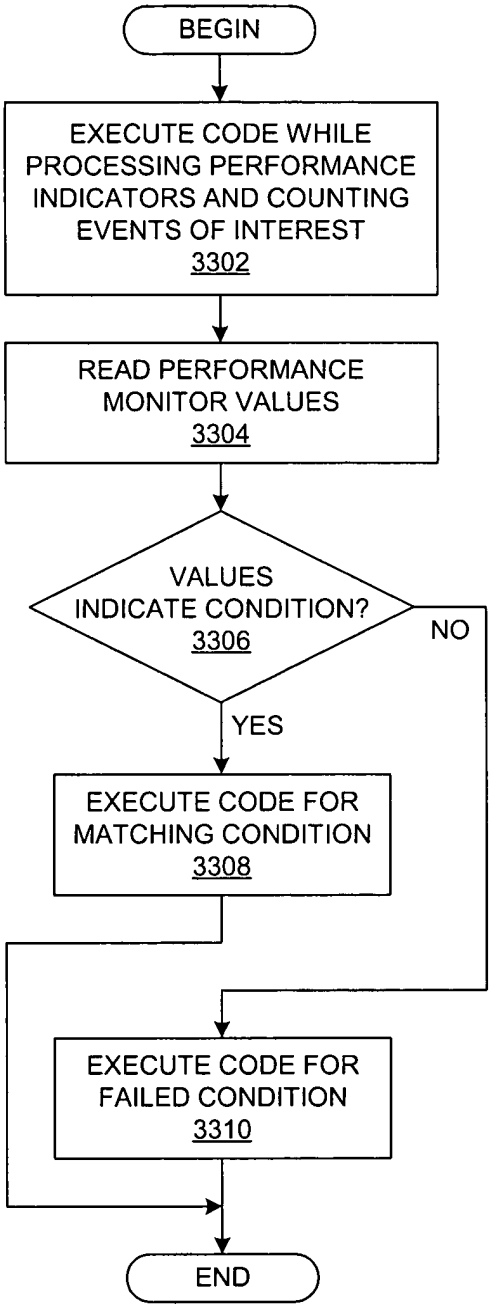

Figure 33

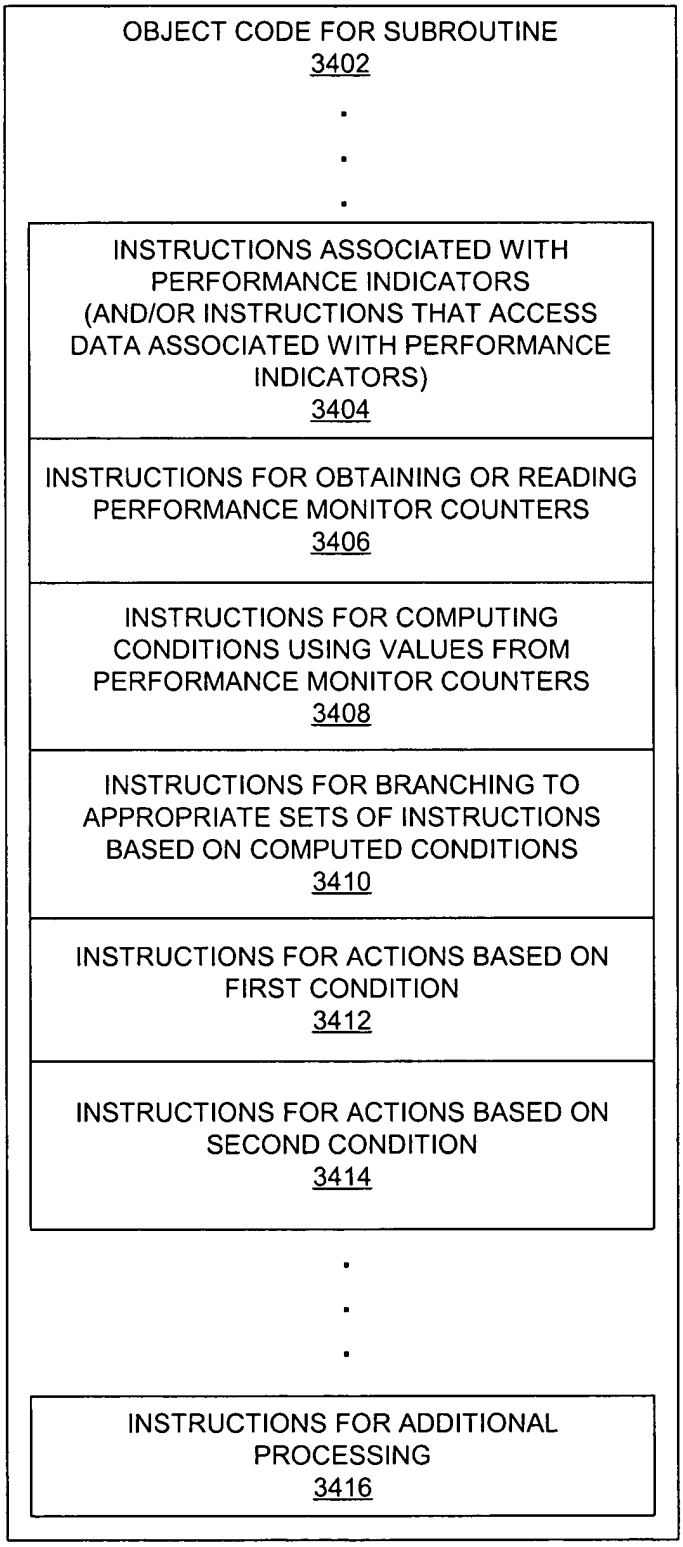
OBJECT CODE FOR SUBROUTINE
3402
INSTRUCTIONS ASSOCIATED WITH PERFORMANCE INDICATORS (AND/OR INSTRUCTIONS THAT ACCESS DATA ASSOCIATED WITH PERFORMANCE INDICATORS)
3404
INSTRUCTIONS FOR OBTAINING OR READING PERFORMANCE MONITOR COUNTERS
3406
INSTRUCTIONS FOR COMPUTING CONDITIONS USING VALUES FROM PERFORMANCE MONITOR COUNTERS
3408
INSTRUCTIONS FOR BRANCHING TO APPROPRIATE SETS OF INSTRUCTIONS BASED ON COMPUTED CONDITIONS
3410
INSTRUCTIONS FOR ACTIONS BASED ON FIRST CONDITION
3412
INSTRUCTIONS FOR ACTIONS BASED ON SECOND CONDITION
3414
INSTRUCTIONS FOR ADDITIONAL PROCESSING
3416

Figure 34

```
3502 ~ Subroutine ProcessData {
                .
                .
                .
3530 ~ ##PerfMon CACHE_MISS, CACHE_HIT;

3504 ~ call GetData(dataStructure, numberDataItems);

3506 ~ call PerfMon_Read( valueA, CACHE_MISS, ~ 3508
                          valueB, CACHE_HIT); ~ 3510
          3512      3514
           )         )
           (         (
3516 ~ if (valueA > valueB) then          3520
                                           )
                                           (
      3518 ~ call ModifyDataStructure(ADJUST, dataStructure);

3522 ~ else if (valueA > (SCALE_CACHE_MISS*numberDataItems)) then

      3524 ~ call ModifyDataStructure(RENEW, dataStructure);
                                          )
                                          (
3528 ~ call SortNewData(dataStructure)   3526
                .
                .
                .
        }
```

Figure 35

```
      Subroutine ProcessData {
                .
                .
                .
3610 ~ ##PerfMon CACHE_MISS, CACHE_HIT;

3602 ~ call GetData(dataStructure, numberDataItems);

3612 ~ ##PerfMon {
      3614 ~ if (CACHE_MISS > CACHE_HIT) then
            3604 ~ call ModifyDataStructure(ADJUST, dataStructure);
      3616 ~ else if (CACHE_MISS > (SCALE_CACHE_MISS*numberDataItems)) then
            3606 ~ call ModifyDataStructure(RENEW, dataStructure);
        }
3608 ~ call SortNewData(dataStructure)
                .
                .
                .
      }
```

Figure 36

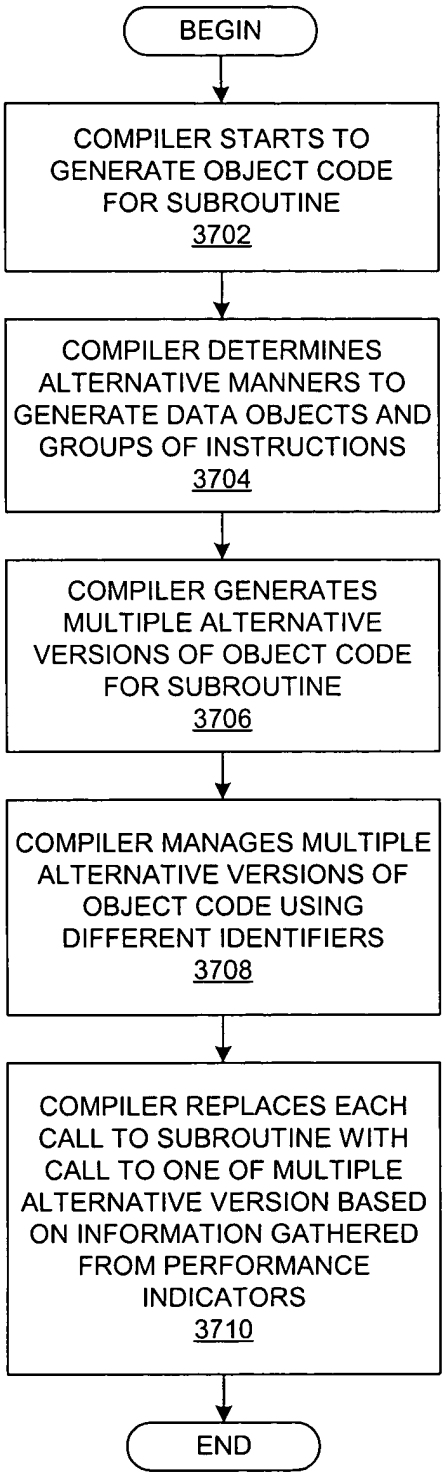
BEGIN
COMPILER STARTS TO GENERATE OBJECT CODE FOR SUBROUTINE 3702
COMPILER DETERMINES ALTERNATIVE MANNERS TO GENERATE DATA OBJECTS AND GROUPS OF INSTRUCTIONS 3704
COMPILER GENERATES MULTIPLE ALTERNATIVE VERSIONS OF OBJECT CODE FOR SUBROUTINE 3706
COMPILER MANAGES MULTIPLE ALTERNATIVE VERSIONS OF OBJECT CODE USING DIFFERENT IDENTIFIERS 3708
COMPILER REPLACES EACH CALL TO SUBROUTINE WITH CALL TO ONE OF MULTIPLE ALTERNATIVE VERSION BASED ON INFORMATION GATHERED FROM PERFORMANCE INDICATORS 3710
END

Figure 37

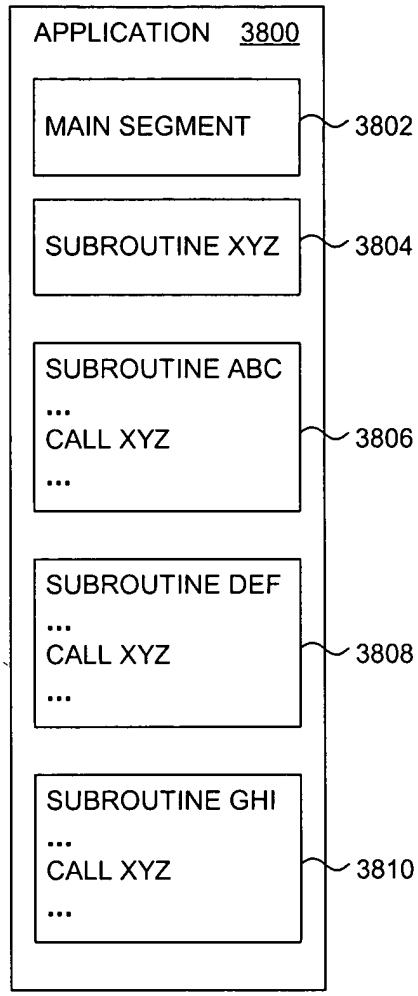

Figure 38

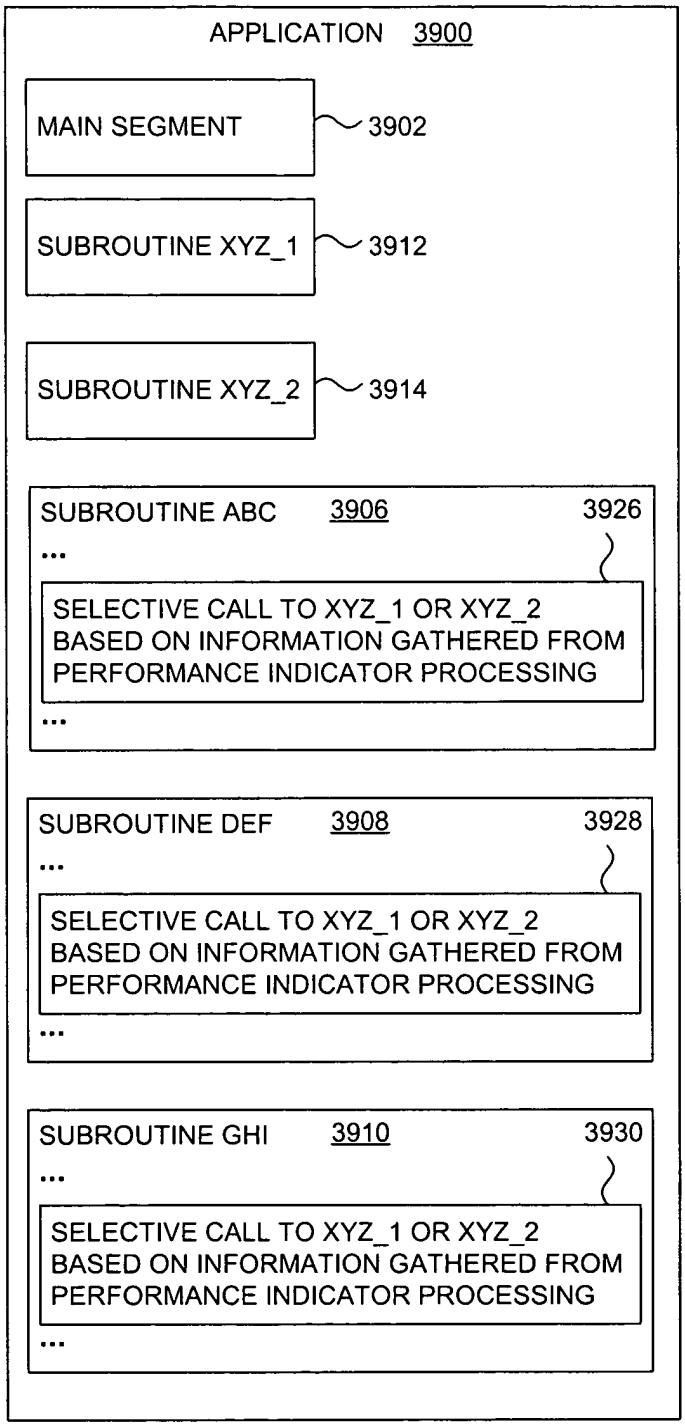
APPLICATION 3900
MAIN SEGMENT
3902
SUBROUTINE XYZ_1
3912
SUBROUTINE XYZ_2
3914
SUBROUTINE ABC 3906
3926
...
SELECTIVE CALL TO XYZ_1 OR XYZ_2 BASED ON INFORMATION GATHERED FROM PERFORMANCE INDICATOR PROCESSING
...
SUBROUTINE DEF 3908
3928
...
SELECTIVE CALL TO XYZ_1 OR XYZ_2 BASED ON INFORMATION GATHERED FROM PERFORMANCE INDICATOR PROCESSING
...
SUBROUTINE GHI 3910
3930
...
SELECTIVE CALL TO XYZ_1 OR XYZ_2 BASED ON INFORMATION GATHERED FROM PERFORMANCE INDICATOR PROCESSING
...

Figure 39

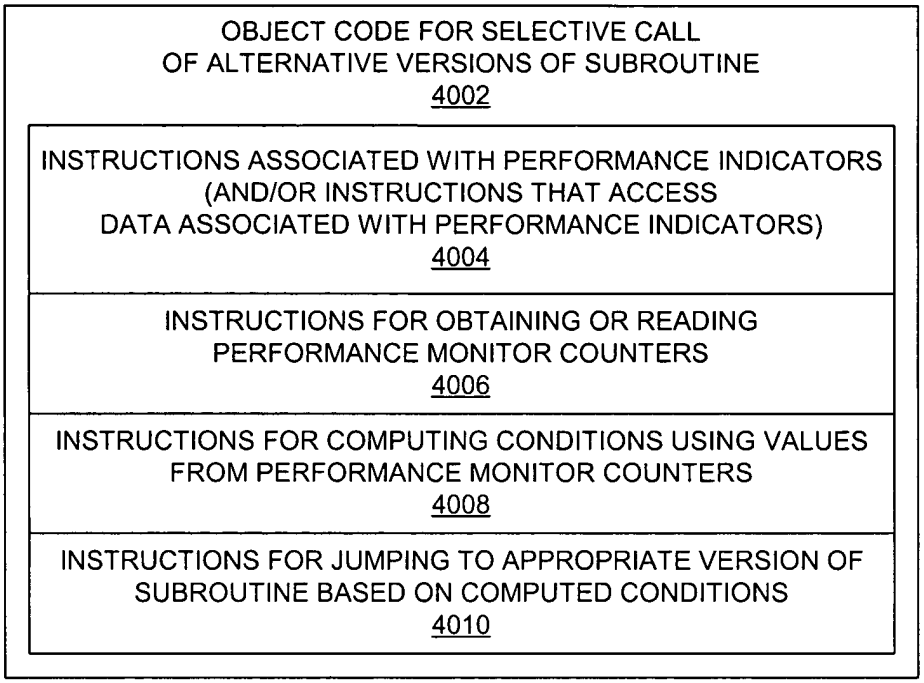

Figure 40

```
                .
                .
                .
4102 ~ ##PerfMon CACHE_MISS on Array_XYZ;

4104 ~ Temp := Array_XYZ[ MAX_ARRAY_SIZE ];

4106 ~ call PerfMon_Read(valueA, CACHE_MISS);

4108 ~ if (valueA = 0) then
       4110 ~ call XYZ_1;
         else
       4112 ~ call XYZ_2;
                .
                .
                .
```

Figure 41

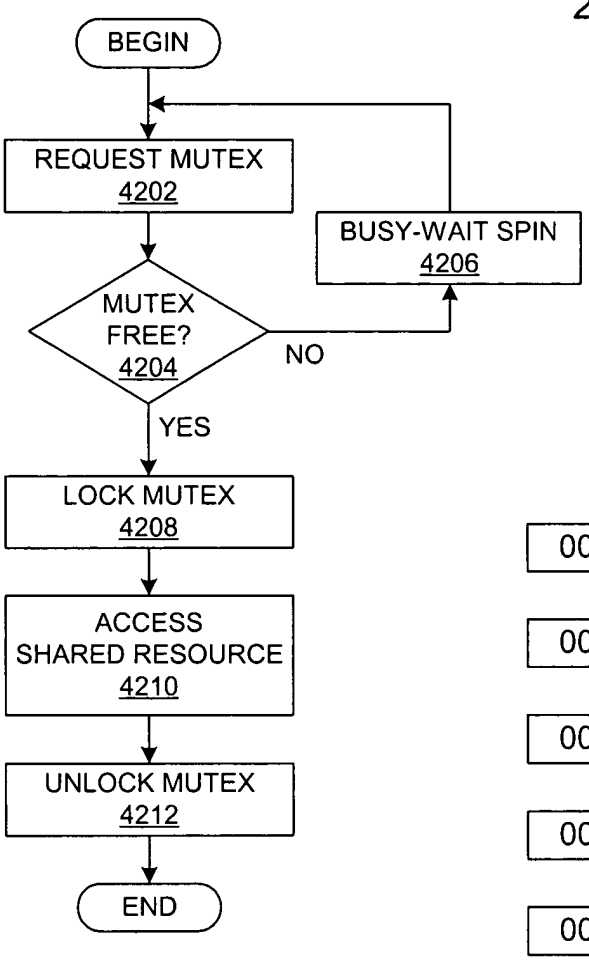

Figure 42
(PRIOR ART)

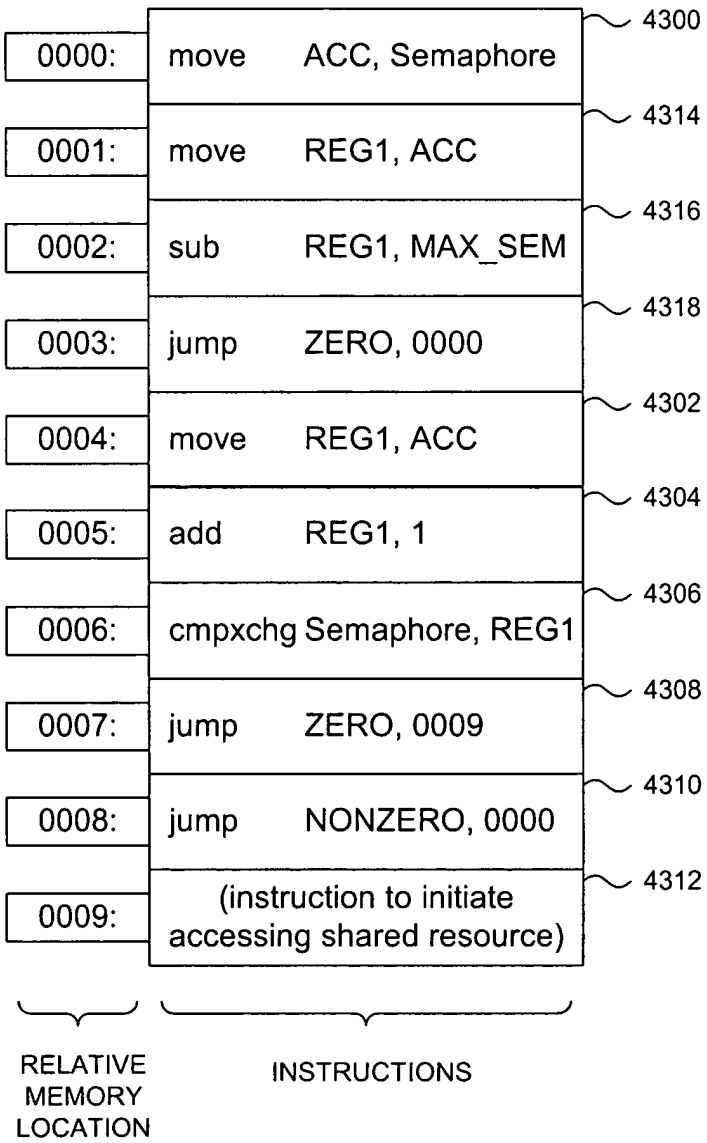

Figure 43
(PRIOR ART)

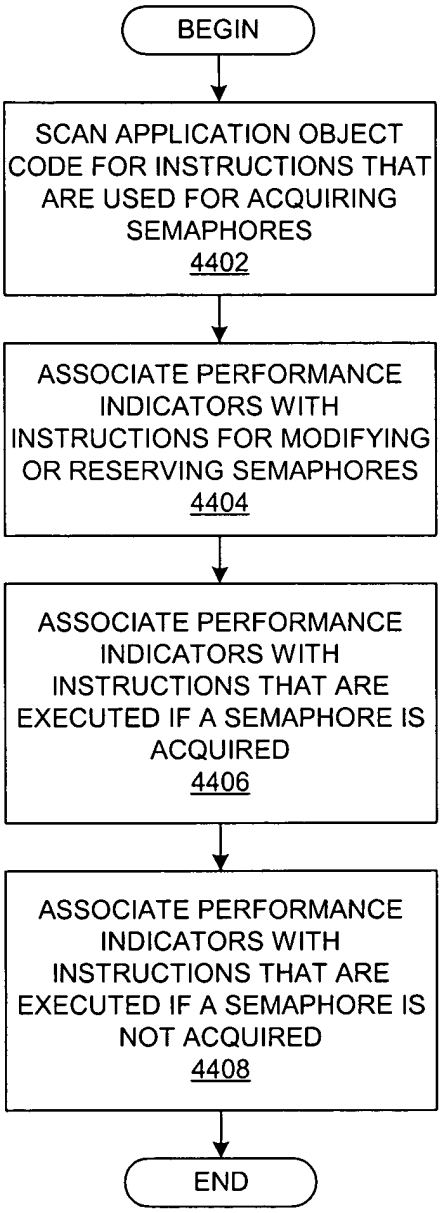
BEGIN
SCAN APPLICATION OBJECT CODE FOR INSTRUCTIONS THAT ARE USED FOR ACQUIRING SEMAPHORES
4402
ASSOCIATE PERFORMANCE INDICATORS WITH INSTRUCTIONS FOR MODIFYING OR RESERVING SEMAPHORES
4404
ASSOCIATE PERFORMANCE INDICATORS WITH INSTRUCTIONS THAT ARE EXECUTED IF A SEMAPHORE IS ACQUIRED
4406
ASSOCIATE PERFORMANCE INDICATORS WITH INSTRUCTIONS THAT ARE EXECUTED IF A SEMAPHORE IS NOT ACQUIRED
4408
END

Figure 44

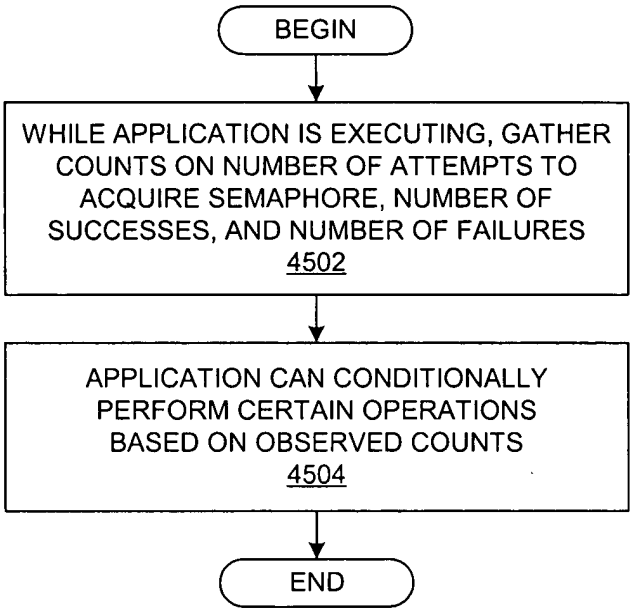
BEGIN
WHILE APPLICATION IS EXECUTING, GATHER COUNTS ON NUMBER OF ATTEMPTS TO ACQUIRE SEMAPHORE, NUMBER OF SUCCESSES, AND NUMBER OF FAILURES
4502
APPLICATION CAN CONDITIONALLY PERFORM CERTAIN OPERATIONS BASED ON OBSERVED COUNTS
4504
END

Figure 45

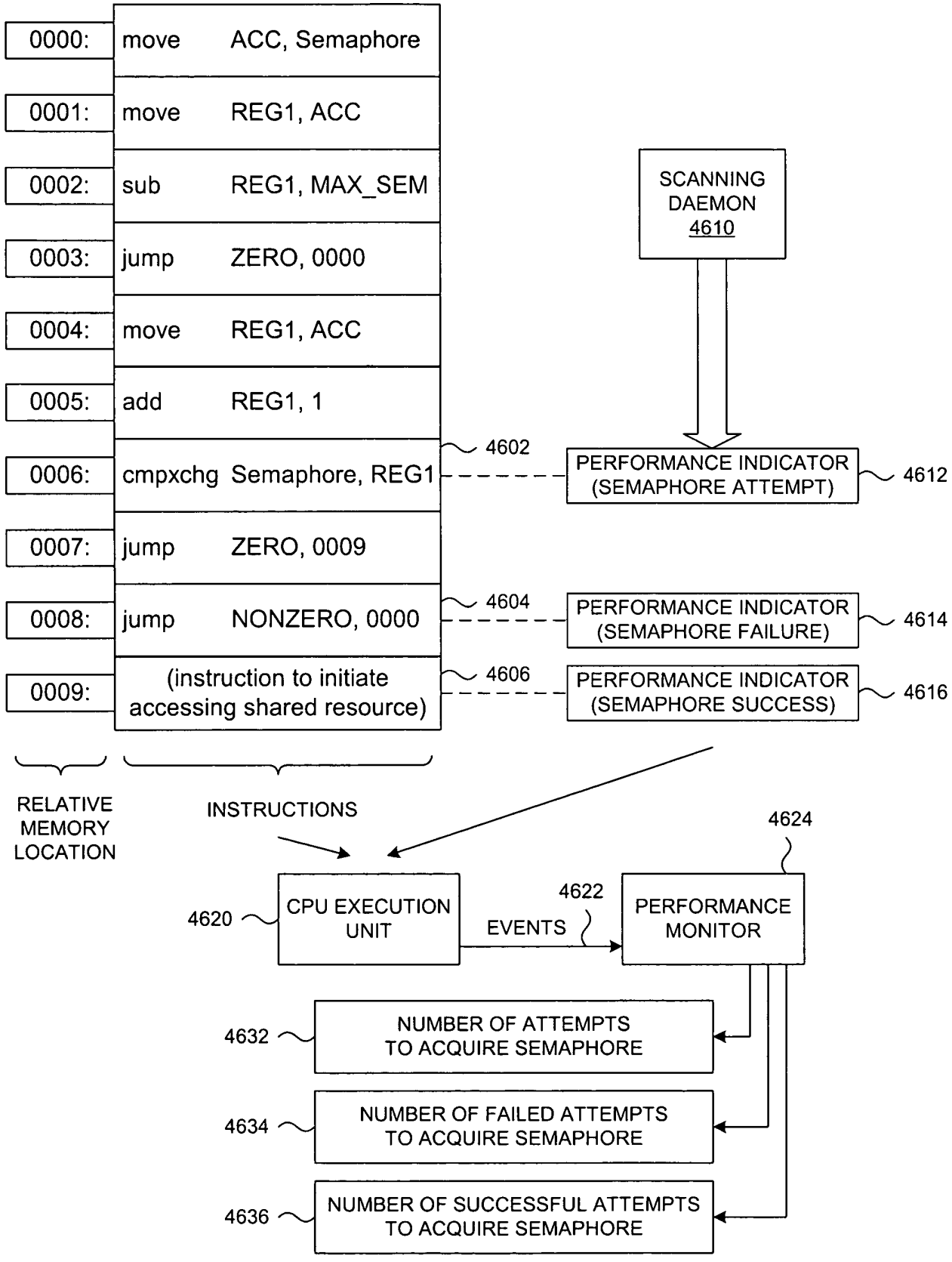
0000: move ACC, Semaphore
0001: move REG1, ACC
0002: sub REG1, MAX_SEM
0003: jump ZERO, 0000
0004: move REG1, ACC
0005: add REG1, 1
0006: cmpxchg Semaphore, REG1
0007: jump ZERO, 0009
0008: jump NONZERO, 0000
0009: (instruction to initiate accessing shared resource)
RELATIVE MEMORY LOCATION
INSTRUCTIONS
SCANNING DAEMON 4610
4602
PERFORMANCE INDICATOR (SEMAPHORE ATTEMPT)
4612
4604
PERFORMANCE INDICATOR (SEMAPHORE FAILURE)
4614
4606
PERFORMANCE INDICATOR (SEMAPHORE SUCCESS)
4616
4620
CPU EXECUTION UNIT
4622
EVENTS
4624
PERFORMANCE MONITOR
4632
NUMBER OF ATTEMPTS TO ACQUIRE SEMAPHORE
4634
NUMBER OF FAILED ATTEMPTS TO ACQUIRE SEMAPHORE
4636
NUMBER OF SUCCESSFUL ATTEMPTS TO ACQUIRE SEMAPHORE

Figure 46

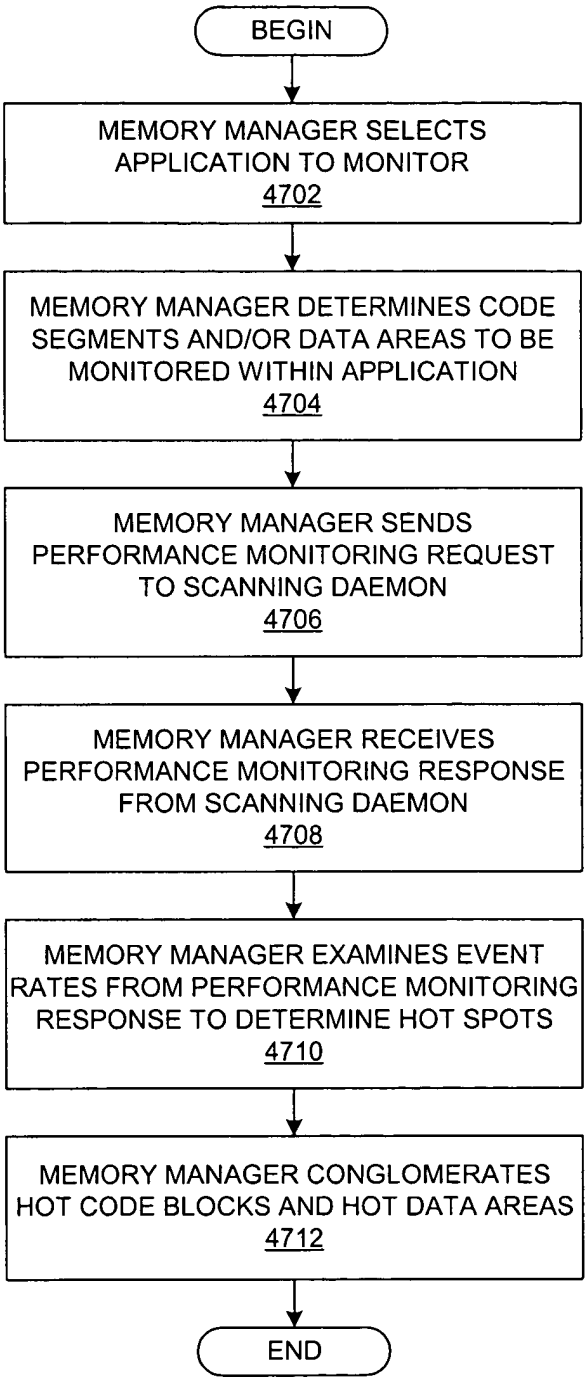
BEGIN
MEMORY MANAGER SELECTS APPLICATION TO MONITOR
4702
MEMORY MANAGER DETERMINES CODE SEGMENTS AND/OR DATA AREAS TO BE MONITORED WITHIN APPLICATION
4704
MEMORY MANAGER SENDS PERFORMANCE MONITORING REQUEST TO SCANNING DAEMON
4706
MEMORY MANAGER RECEIVES PERFORMANCE MONITORING RESPONSE FROM SCANNING DAEMON
4708
MEMORY MANAGER EXAMINES EVENT RATES FROM PERFORMANCE MONITORING RESPONSE TO DETERMINE HOT SPOTS
4710
MEMORY MANAGER CONGLOMERATES HOT CODE BLOCKS AND HOT DATA AREAS
4712
END

Figure 47

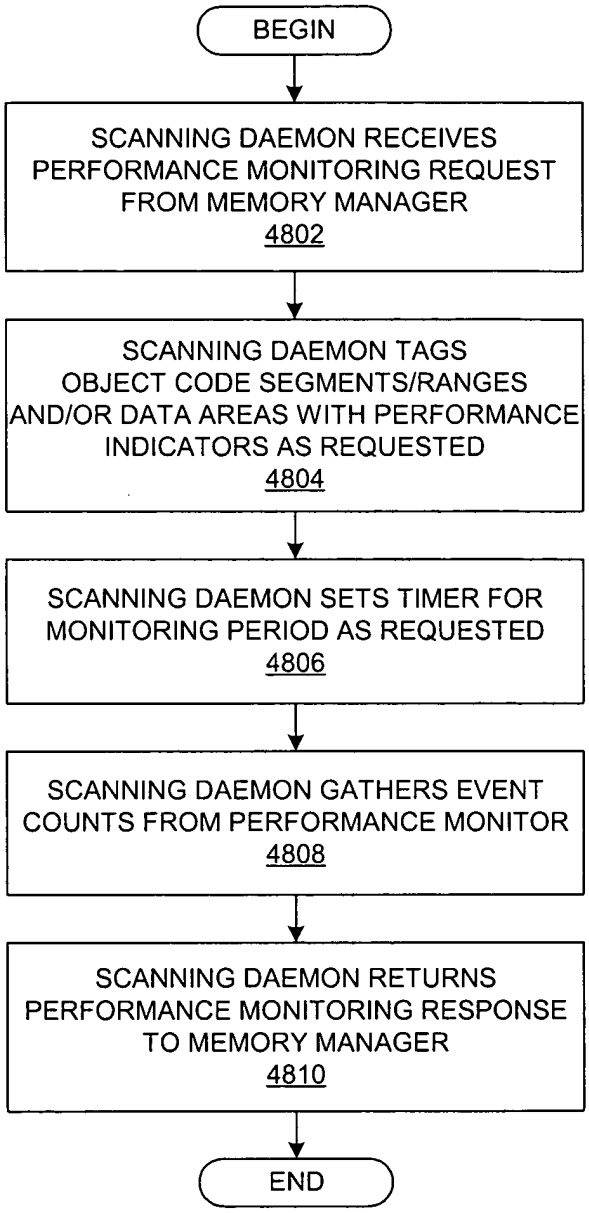
BEGIN
SCANNING DAEMON RECEIVES PERFORMANCE MONITORING REQUEST FROM MEMORY MANAGER
4802
SCANNING DAEMON TAGS OBJECT CODE SEGMENTS/RANGES AND/OR DATA AREAS WITH PERFORMANCE INDICATORS AS REQUESTED
4804
SCANNING DAEMON SETS TIMER FOR MONITORING PERIOD AS REQUESTED
4806
SCANNING DAEMON GATHERS EVENT COUNTS FROM PERFORMANCE MONITOR
4808
SCANNING DAEMON RETURNS PERFORMANCE MONITORING RESPONSE TO MEMORY MANAGER
4810
END

Figure 48

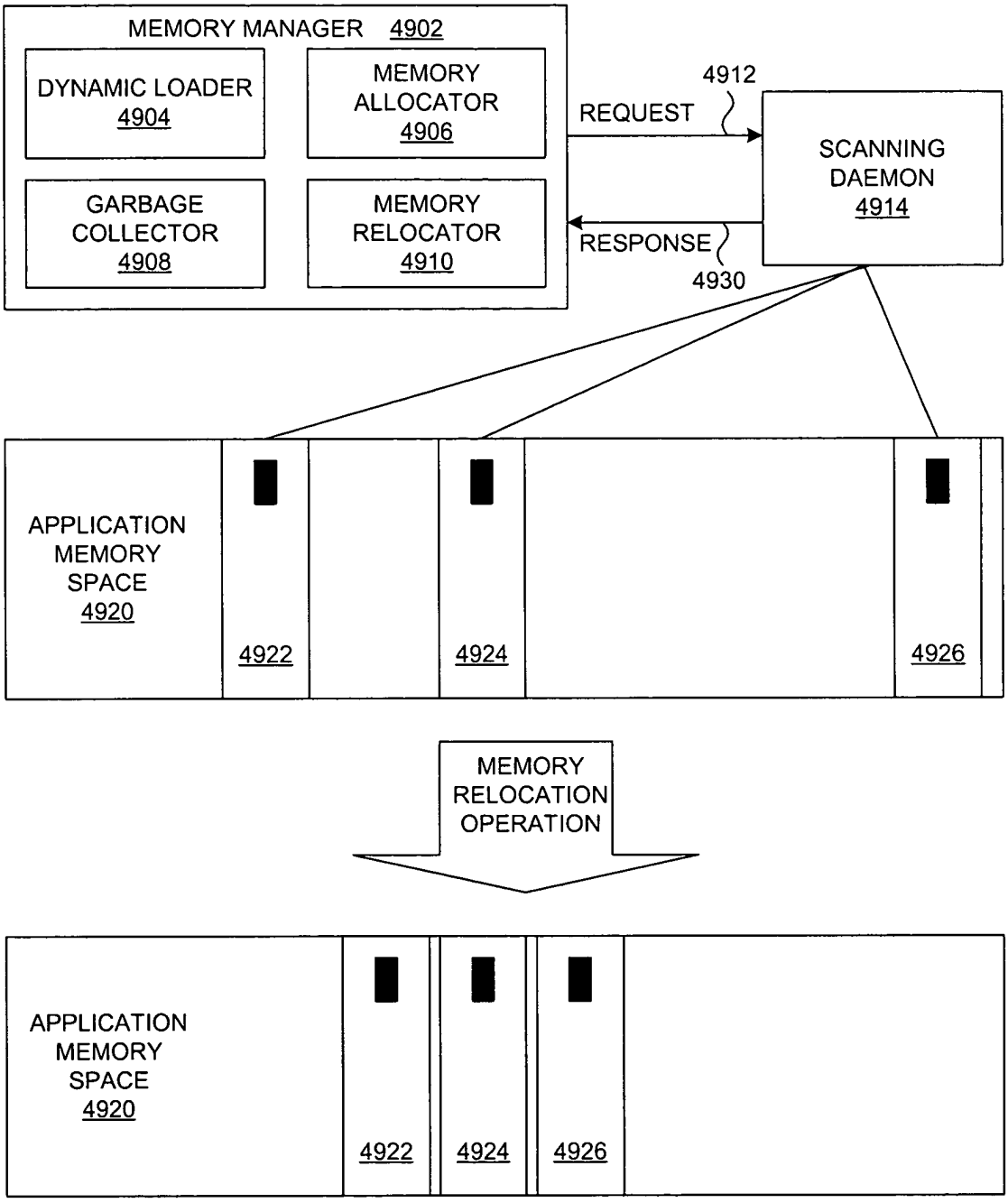
MEMORY MANAGER 4902
DYNAMIC LOADER 4904
MEMORY ALLOCATOR 4906
GARBAGE COLLECTOR 4908
MEMORY RELOCATOR 4910
4912
REQUEST
RESPONSE
4930
SCANNING DAEMON 4914
APPLICATION MEMORY SPACE 4920
4922
4924
4926
MEMORY RELOCATION OPERATION
APPLICATION MEMORY SPACE 4920
4922
4924
4926

Figure 49